基于梦境粒子群优化的类集成测试序列生成方法

张悦宁[1] 姜淑娟[1] 张艳梅[1,2]

(中国矿业大学计算机科学与技术学院矿山数字化教育部工程研究中心 江苏 徐州 221116)[1]

(桂林电子科技大学广西可信软件重点实验室 广西 桂林 541004)[2]

摘 要 类集成测试序列的确定是面向对象类集成测试技术中的一个重要课题。合理的类集成测试序列可以降低为其构造测试桩的总体复杂度,从而减小测试代价。针对粒子群优化算法容易早熟的缺陷,文中提出一种基于梦境粒子群优化算法的类集成测试序列生成方法。首先把每个类集成测试序列映射为一维空间中的一个粒子,然后将粒子看作有做梦能力的个体。每个迭代周期分为白天和夜间两个阶段,在白天阶段粒子正常移动,而在夜间阶段粒子根据各自的做梦能力扭曲当前位置。如此,粒子有机会在当前位置附近进行搜索,使得算法减缓收敛速度,避免过早陷入局部最优。实验结果表明,多数情况下该方法可以得到测试代价更小的类集成测试序列。

关键词 测试序列,集成测试,测试代价,梦境粒子群优化算法,局部最优

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.02.025

Approach for Generating Class Integration Test Sequence Based on Dream Particle Swarm Optimization Algorithm

ZHANG Yue-ning[1] JIANG Shu-juan[1] ZHANG Yan-mei[1,2]

(Mine Digitization Engineering Research Center of the Ministry of Education, School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu 221116, China)[1]

(Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, Guangxi 541004, China)[2]

Abstract Determination of class integration test sequence is an important topic in object-oriented software integration testing. Reasonable class integration test sequence can reduce the overall complexity of test stub, and then reduce test cost. For particle swarm optimization algorithm, it is easy to be precocious. So a class integration test sequence determination method based on dream particle swarm optimization algorithm was proposed in this paper. First, each sequence is taken as a particle in one dimensional space. Then, every particle is considered to be a dreamer. Each iteration cycle is divided into two phases: day and night. In the daytime, particles move to new locations, and during the night, they contort the locations gained at day phase according to dreaming ability. In this way, particle has the opportunity to search near the current location, so that the algorithm can converge slowly and avoid falling into local optimum too early. The experimental results show that the proposed approach takes a lower test cost in most cases.

Keywords Test sequence, Integration testing, Test cost, Dream particle swarm optimization algorithm, Local optimum

1 引言

类集成测试序列的确定是面向对象软件集成测试技术研究中的一个重要内容,也是难点所在。对于不同的类测试序列,其测试代价往往也不同,因此确定合理的类集成测试序列对减小测试代价、降低测试成本具有重要意义。现有的确定类集成测试序列的方法中,基于图论的方法和基于搜索的方法较多,此外还有一些结合切片技术和复杂网络理论的方法。

基于图论的方法最早出现,理论成熟。该方法的复杂度与类的数量及类间关系直接相关,在小型系统上尚可实现,但难以应对实际中规模较大的系统。基于搜索的方法较好地突破了系统规模的限制,其基本思想是:首先确定一个初始种群,种群中的每个个体代表一个类测试序列;然后在满足一定条件(如尽可能地降低测试桩的复杂度)的前提下,通过进化操作使种群迭代;最后生成最优个体,进而得到最优的类集成测试序列。代表方法有基于遗传算法(Genetic Algorithm, GA)[1-2]的方法、基于随机迭代算法(Random Iterative Algorithm, RIA)[3]的方法、基于粒子群优化算法(Particle Swarm

到稿日期:2018-08-11 返修日期:2018-10-16 本文受国家自然科学基金(61673384,61502497),广西可信软件重点实验室开放课题(kx201530)资助。

张悦宁(1993-),男,硕士生,主要研究领域为软件测试、类集成测试序列生成等,E-mail:ynzhang@cumt.edu.cn;姜淑娟(1966-),女,博士,教授,博士生导师,CCF高级会员,主要研究领域为编译技术、软件工程等,E-mail:shjjiang@cumt.edu.cn(通信作者);张艳梅(1982-),女,博士,副教授,CCF会员,主要研究领域为软件分析与测试。

Optimization,PSO)[4]的方法等。

PSO 通过模拟鸟群的飞行行为,将每只飞行的鸟看作一个粒子,根据种群中各粒子之间的相互合作和学习寻找最优位置。该算法简单有效,鲁棒性强且收敛速度快,在多数情况下能够得到比 GA 方法和 RIA 方法的总体复杂度更低的类集成测试序列[4]。但是粒子群优化算法存在过早收敛的缺陷,在进化过程中所有粒子都向种群当前的最优位置靠近,导致进化开始不久粒子就大量聚集,种群多样性降低,算法过早陷入局部最优而错过更好的解。

鉴于此,本文提出一种基于梦境粒子群优化(Dream Particle Swarm Optimization,DPSO)算法[5]的类集成测试序列生成方法。在 DPSO 中,粒子被认为是有做梦能力的个体,每次迭代分为两个阶段——白天和黑夜。白天阶段,粒子从前一晚的位置正常移动。夜间阶段,粒子进入梦境,根据各自的做梦能力扭曲自身的当前位置。由此,粒子不会直接飞向当前的局部最优位置,而是在附近位置进行搜索,因此有机会找到更优的位置。同时,随着迭代次数的增加,粒子的做梦能力将逐渐减小,从而保证算法最终收敛。实验结果表明,多数情况下本文方法生成的类集成测试序列的测试代价更小。

2 相关工作

目前类集成测试序列的确定方法大致分为 4 种:基于图论的方法、基于搜索的方法、基于切片技术的方法以及结合复杂网络理论的方法。

2.1 基于图论的方法

对于无环图,Kung 等[6-7]基于对象关系图(ORD)[8]通过逆向拓扑排序生成类测试序列;对于有环图,需要先删除关联边消除环路,得到无环图后再进行逆向拓扑排序。

之后,研究者提出了多种方案来为有环图的边赋值,优先删除权值最大的边直到没有环路为止。Tai 等[9]将各关联边的起点入度和终点出度之和作为权值;Le 等[10]采用了一种基于测试依赖图(TDG)模型的方法。该测试依赖图由类和方法之间的测试依赖关系构成,每个节点被赋予一个权值,然后删除具有最大权重的节点的入边,直至消除所有环路。Briand 等[11-12]明确区分了特效桩和普通桩,利用 Tarjan 算法[13]对 ORD 进行深度优先遍历,划分强联通分量(SCCs),然后打破强连通分量的环路。

上述方法只考虑了静态依赖,具有局限性,因为类间普遍存在的动态依赖关系同样会形成环路。张艳梅等[14-15]考虑了动态依赖对确定类集成测试序列的影响。

以上研究的目标都是减少测试桩的数量,然而构造不同测试桩的代价往往不同,因此测试桩的数量越少并不能说明该测试序列的总体测试代价就越小。相比于测试桩数量,测试桩的复杂度更能准确地衡量一个类集成测试序列的测试代价。Briand 等[11]从类间方法调用和属性依赖的角度度量测试代价,提出了方法复杂度和属性复杂度的概念。后来学者们在 Briand 的基础上展开了进一步的研究。

姜淑娟等[16]对测试桩复杂度进行了新的耦合度量,提出了一种打破环路的图论算法。此外,Hewett 等[17]提出了一种自底向上增量式的图论算法,通过分析类间关系每次选出下一个待测类,并在衡量测试代价时考虑了特效桩和普通桩。Zhang 等[18]采用基于奖惩机制的多级反馈策略生成类集成测试序列,在缩短测试时间的同时降低了测试代价。

2.2 基于搜索的方法

根据多目标优化模型的不同,基于搜索的方法分为基于线性加权模型的方法和基于帕累托最优模型的方法。

基于线性加权模型的方法应用得最广。Briand 等[1-2]和 Hanh 等[19]均设计了遗传算法生成类集成测试序列的方案。Borner 等[20]将属性复杂度、方法复杂度和测试焦点作为优化目标,利用模拟退火算法生成类集成测试序列。Wang 等[3]对遗传算法、模拟退火算法等进行了改进,依据测试桩代价最小化的原则生成类集成测试序列。张艳梅等[4]使用粒子群优化算法,将每个类测试序列看作一个粒子,迭代完成后再通过最优粒子位置获得对应的类集成测试序列。

基于帕累托模型的方法众多。Cabral 等[21]利用基于帕累托模型的蚁群算法生成类集成测试序列。Vergilio 等[22]根据禁忌搜索的思想,利用禁忌列表保存生成过的类集成测试序列以避免重复搜索。Assunção 等[23]比较了 3 种常见的基于帕累托模型的多目标优化算法,即 NSGA-II[24],SPEA2[25]和 PAES[26],并将其用于解决类集成测试序列的生成问题。

近年来,超启发式算法(Hyper-Heuristic)受到了研究者的重视。超启发式算法提供某种高层策略,通过管理或操纵一组底层启发式算法获得新的启发式算法。Guizzo 等[27-28]利用超启发式算法为遗传算法在每一次进化中提供交叉和变异的多种可选组合,最终得到了满意的类集成测试序列。

2.3 基于切片技术的方法

Jaroenpiboonkit 等[29]将面向对象切片技术用于确定类集成测试序列。该方法不作用在类级别,而是在方法层面按照依赖关系对类进行切分,从而在无需构建测试桩的情况下打破环路,在得到无环图后进行逆向拓扑排序以得到类测试序列。刘颖莲[30]提出了一种基于切片技术的面向对象软件集成测试策略,根据类间依赖关系图中节点的权值进行切片,再为每个切片计算权值并根据权值确定测试顺序。

2.4 结合复杂网络理论的方法

赵玉丽等[31]根据类间调用关系及类自身的复杂度,定义了类节点重要度的衡量指标——复杂因子和影响因子,并以此为据计算节点的重要性,在保证测试桩复杂度较小的同时优先测试重要性高的类。之后,王莹等[32]依据“复杂类的错误传播倾向更高”的思想设计了度量类重要性的标准——复杂性和影响力,将类错误倾向指数作为类的重要性初值,再利用 Webmining 中的 HITS 算法[33]计算类的重要性,最后根据类的重要性和测试桩复杂度确定类集成测试序列。

3 基于 DPSO 的类集成测试序列生成

为了解决粒子群算法易早熟的问题,本文提出一种基于梦境粒子群(DPSO)算法的类集成测试序列生成方法。

3.1 相关概念

定义 1(测试桩[19]) 给定两个类 i 和 j,其中 i 依赖于 j。在集成测试过程中,如果在测试 i 时还未测试 j,则需要构建组件来模拟 j 的功能以保证 i 的测试顺利进行。该模拟组件

就称为类 j 的测试桩。

定义 2（属性复杂度[11]）　若类 i 声明的属性引用了类 j 的属性，则称类 i 存在对类 j 的属性依赖。此时，类 i 对类 j 的属性依赖值称为属性复杂度，记为 $A(i,j)$。

定义 3（方法复杂度[11]）　若类 i 在方法中调用了类 j 定义的方法，则称类 i 存在对类 j 的方法依赖。此时，类 i 对类 j 的方法依赖值称为方法复杂度，记为 $M(i,j)$。

3.2　测试桩复杂度

采用耦合度量的方式衡量类测试序列的代价，类间耦合信息包含两部分[1]：属性依赖和方法依赖。

Briand 等[1]和张艳梅等[4]打破环路时都不允许删除类间的强依赖关系（继承、组合、聚集），为了便于对比，本文在粒子迭代过程中也不允许删除类间的强依赖关系。

测试桩复杂度可以衡量构造一个测试桩的代价，复杂度越高，其代价就越大。对于存在依赖关系的两个类 i 和 j，如果需要为类 j 构造一个测试桩，则其复杂度 $SCplx(i,j)$ 可由式(1)计算[1]：

$$SCplx(i,j)=\sqrt{W_A\cdot\overline{A(i,j)}^2+W_M\cdot\overline{M(i,j)}^2} \tag{1}$$

其中，W_A 和 W_M 分别为属性复杂度和方法复杂度的权重，取值范围为[0,1]，且 $W_A+W_M=1$；$\overline{A(i,j)}$ 和 $\overline{M(i,j)}$ 分别表示属性复杂度和方法复杂度标准化的结果。文献[2]通过实验证明了属性复杂度和方法复杂度在具有相同权重时的效果最佳，因此本文采用同样的设置，即 $W_A=W_M=0.5$。

总体复杂度指一个测试序列构造的所有测试桩的代价之和。对于测试序列 O，其测试桩的总体复杂度为[1]：

$$OCplx(o)=\sum_{i=1,j=1}^{n}SCplx(i,j) \tag{2}$$

3.3　梦境介绍

研究表明，自然界中的高级生命都遵循白天-黑夜两个阶段的活动规律。以人类为例，白天进行学习、工作等各项生理活动，而在夜晚的睡眠期间主要进行心理活动，也就是做梦。自古以来就有对梦境的记载，最早人们认为梦与神有关，而在后来的科学研究中，关于梦的理论才与大脑的功能联系起来。睡眠实验室技术的发展揭开了梦境领域研究的新篇章。在生理因素上，普遍的观点是睡眠周期分为快速眼动期（REM）和非快速眼动期（NREM），在 REM 期间大脑对新形成的记忆进行整合分析[34]。在人类梦境的研究中，Hobson 等提出了一种两阶段模型[35]。他们认为，人在睡眠时从脑干神经释放出的信号会刺激大脑皮层区域，此时激活了做梦者过去的记忆和经验，人开始做梦。此外，在人类生命的不同阶段，REM 期的长短也是不同的，基本呈高斯分布。例如：新生婴儿需要 16 个小时的睡眠，其中半数是快速眼动睡眠；而年轻人大概有两个小时的快速眼动睡眠；老人则只需 30 分钟的快速眼动睡眠。

3.4　本文方法的框架

基于 DPSO 的类集成测试序列生成方法的框架如图 1 所示，该框架共有 3 部分：静态分析、初始化和基于 DPSO 的类集成测试序列生成。

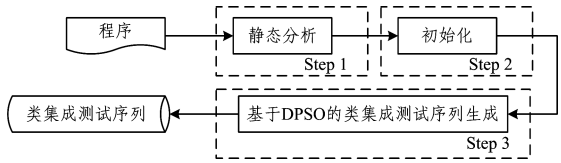


图 1　DPSO 方法的框架

Fig. 1　Framework of DPSO

(1)静态分析：获取程序的类间依赖关系、成员变量和成员方法。计算属性复杂度和方法复杂度，为计算适应度函数值做准备。

(2)初始化：每个类测试序列被映射为一维空间中的一个粒子[4]，映射方法如算法 1 所示。然后，随机产生一定数量的粒子作为初始种群。

(3)基于 DPSO 的类集成测试序列生成：算法根据第(1)部分得到的数据计算每个粒子在迭代中的适应度值，经过迭代得到适应度值最优的粒子并记录其位置，最后将该位置逆映射为对应的类集成测试序列。

算法 1　类测试序列映射为粒子的算法

输入：初始类序列 LIST_Seq，程序类链表 LIST_Cla（表中的类按照其依赖数降序排列）

输出：粒子位置 LOC

```
1. BEGIN
2. LOC=0;
3. n=LIST_Cla.size();/* n 为待测程序包含类的个数 */
4. FOR (int i=0;i<n;i++) /* i 为第 i 个待测类 */
5.    class=LIST_Seq.get(i);
6.    index=LIST_Cla.getIndex(class);
7.    LOC=LOC+index*(n-i-1)!;
8. ENDFOR
9. END
```

算法 1 中，第 2 行指定粒子的初始位置为 0；第 3 行得到待测程序中类的总数；第 4－8 行确定类序列对应的粒子位置，其中第 5 行取出 LIST_Seq 中的 1 个类，第 6 行获得该类在链表 LIST_Cla 中的坐标，最后当循环结束时根据第 7 行的公式即可得到类序列映射的粒子位置。

例 1　给定一个程序，其包含 Student，Teacher，School 3 个类，类间的依赖关系为方法依赖和属性依赖，依赖数如表 1 所列。

表 1　类间依赖数

Table 1　Dependencies between classes

	Student	Teacher	School
Student		3	1
Teacher	2		1
School	1	1	

在表 1 中，数字表示纵坐标中的类对横坐标中的类的依赖数。例如，由表第二行可知 Student 类对 Teacher 类的依赖数为 3，对 School 类的依赖数为 1，故 Student 类总的依赖数为 4(3+1)。同理，Teacher 类总的依赖数为 3(2+1)，School 类总的依赖数为 2(1+1)。将类序列映射为粒子位置，首先按类间依赖数降序排列得到链表 LIST_Cla＝{Student，Teacher，School}。若给定一个类序列 LIST_Seq＝{School，Student，Teacher}，根据算法 1，该类序列对应的粒子位置为：

$$LOC(LIST\_Seq)=\sum_{i=0}^{2}[index(LIST\_Cla(i))\times(3-i-1)!]=4$$

3.5 DPSO 算法的原理

首先,使用高斯分布初始化粒子的做梦能力。$A_i(x_1,x_2,\cdots,x_D)(i\in[1,n])$代表做梦能力[5],其中 i 是粒子的编号,n 表示粒子群的个体数量,D 表示维度。计算公式如下:

$$A_i(x_d)=\frac{1}{\sqrt{2\pi}}\exp(-\frac{i^2}{2})\times\omega_f\times\frac{MAX-k}{MAX}\times\eta \tag{3}$$

其中,η 为梦境参数,表示粒子对当前位置的扭曲程度;k 表示当前的迭代次数;MAX 表示最大迭代次数;$A(x_d)$将会在迭代过程中线性减小到 0;权重 ω_f 表示粒子适应度值对梦境的影响,由适应度函数值标准化得到。在类集成测试序列生成过程中,适应度值代表了测试序列的总体复杂度,因此适应度值较小(较优)的粒子将较少受到梦境的影响,以防粒子错过较优的位置。令 $\omega_{f(i)}(i\in[1,n])$表示每次迭代中第 i 个粒子的权重,其中,n 为粒子个数,p_g 为本次迭代的全局最优解,则 $\omega_{f(i)}$ 如式(4)所示:

$$\omega_{f(i)}=\frac{fitness_i-fitness_{\min}}{fitness_{\max}-fitness_{\min}}=\frac{fitness_i-p_g}{fitness_{\max}-p_g} \tag{4}$$

对于一个类集成测试序列,把 $Ocplx(o)$的计算结果作为测试桩的总体复杂度。DPSO 算法求解类集成测试序列问题时,通过适应度函数评价粒子的优劣,因此将总体复杂度公式作为梦境粒子群算法的适应度函数,如式(5)所示[1]:

$$fitness=Ocplx(o)=\sum_{i=1,j=1}^{n}SCplx(i,j) \tag{5}$$

在夜间阶段,位置信息将被 $A(x_d)$扭曲,$x_i(t)(i\in[1,n])$表示粒子的位置。位置的更新公式如下:

$$\Delta x=rand(-1,1)\times A_i(x_d) \tag{6}$$

$$x_i(t+1)=x_i(t)+\Delta x+v_i(t+1) \tag{7}$$

对于每个粒子,其速度和位置的更新公式如式(8)、式(9)所示:

$$v_i(t+1)=\omega v_i(t)+c_1r_1(t)(p_l(t)-x_i(t))+c_2r_2(t)(p_g(t)-x_i(t)) \tag{8}$$

$$x_i(t+1)=x_i(t)+rand(-1,1)\times\frac{1}{\sqrt{2\pi\eta}}\exp(-\frac{i^2}{2})\times\omega_{f(i)}\times\frac{MAX-k}{MAX}+v_i(t+1) \tag{9}$$

式(8)中,c_1 和 c_2 均为常数;r_1 和 r_2 是(0,1)之间的随机数;p_l表示个体最优解;p_g 表示全局最优解;ω 为惯性权重,取值范围在(0.4,0.9)之间且随迭代次数递减。

3.6 算法流程

DPSO 的实现步骤如下:

Step1 初始化粒子的位置和速度;

Step2 根据当前位置计算每个粒子的适应度值;

Step3 计算并记录个体最优解和全局最优解;

Step4 根据式(4)标准化适应度值,并计算权重;

Step5 根据式(3)计算粒子的做梦能力;

Step6 根据式(8)更新粒子速度;

Step7 根据式(6)得到扭曲距离差值;

Step8 根据式(9)扭曲粒子到新的位置;

Step9 判断是否满足终止条件,若满足则停止算法并输出结果,否则执行 Step2。

4 实验

为了检验方法的有效性,选取开源程序作为实验对象,提出 3 个研究问题,收集实验结果并加以分析。实验环境为:Windows 10 操作系统,酷睿 i5 双核处理器(2.6 GHz),物理内存 4.0 GB。

4.1 实验对象

选取 ATM,ANT,SPM,DNS,BCEL,JBoss,JHotDraw 和 MyBatis 8 个系统开展实验。ATM[2] 是一个自动取款机模拟系统。ANT[2] 是基于 Java 平台的开源代码,是 Jakarta 程序中的一部分。SPM[2] 是一个保安巡逻监控系统。DNS[2] 是提供网络域名服务的系统。BCEL[2] 是一个方便用户对 Java 类文件进行分析、创建和操纵的系统。JBoss 是一个开源的应用服务器系统。JHotDraw 是一个二维的图形编辑框架。MaBatis 是一个基于 Java 的持久层框架。详细信息如表 2 所列,第 2-5 列分别给出了实验程序的语言、类数、依赖数和代码行数。

表 2 系统详细信息

Table 2 Information of subject programs

系统	语言	类数	依赖数	代码行数
ATM	Java	21	67	1390
ANT	Java	25	83	4093
SPM	Java	19	72	1198
DNS	Java	61	276	6710
BCEL	Java	45	289	3033
JBoss[1)]	Java	150	367	14896
JHotDraw[2)]	Java	411	809	78150
MyBatis[3)]	Java	428	1271	19554

类的数量和代码行数决定了系统的规模,由表 2 中的信息可以看出 8 个系统的规模各不相同。前 5 个系统的规模较小,类间依赖数较少,而后 3 个系统的规模较大,依赖数较多,由此可以检验方法在不同规模的系统下的效果。

4.2 实验对比

设计 3 组实验,选取基于遗传算法(GA)[2]、基于随机迭代算法(RIA)[3] 和基于粒子群优化算法(PSO)[4] 的类集成测试序列生成方法,分别从测试桩复杂度、收敛速度和运行时间 3 个方面与本文方法进行对比分析,研究的问题如下:

(1)本文方法是否可以找到测试桩复杂度更小的类集成测试序列?

(2)相比 PSO 方法,本文方法是否可以避免粒子过早陷入局部最优?

(3)本文方法在运行时间上是否具有可行性?

1) http://sir.unl.edu/content/bios/jboss.php package used:org.jboss.management of jboss (v0)

2) http://sourceforge.net/projects/jhotdraw package used:org.jhotdraw.draw of JHotDraw (v7.5.1)

3) http://code.google.com/p/mybatis MyBatis (v3.0.2)

4.3　参数设置

梦境粒子群算法除了设置常规参数，还需设置梦境参数 η。其中，常规参数包括初始种群的规模、迭代次数、速度范围、W_A 和 W_M 等。为了便于对比实验结果，本文参照文献[4]进行参数设置，如表 3 所列。

表 3　常规参数设置

Table 3　Conventional parameters setting

系统	初始种群	迭代次数	W_A	W_M	速度范围
ATM，ANT，SPM，BCEL，DNS，	100	500	0.5	0.5	$[0,n!/n]$
JBoss	300	500	0.5	0.5	$[0,n!/n]$
JHotDraw，MyBatis	1000	500	0.5	0.5	$[0,n!/n]$

注：n 为待测系统中类的个数

由式(3)和式(6)可知，粒子的偏移值 Δx 是由梦境参数 η 乘以一个[0，1]之间的数($\frac{1}{\sqrt{2\pi}}\exp(-\frac{i^2}{2})\times\omega_f\times\frac{MAX-k}{MAX}$)，再乘以一个[−1，1]之间的随机数得到的。为了保证在式(7)中 Δx 的数量级和 $x_i(t)$ 的数量级接近，使得 Δx 可以和 $v_i(t+1)$ 一样对 $x_i(t)$ 产生明显的偏移作用，增强算法效力，本文将梦境参数 η 设置为粒子的最大速度 V_{MAX}，即 $\eta=V_{MAX}=n!/n$。

4.4　实验结果及分析

考虑到算法的随机性，所有实验数据均为重复执行 50 次实验所得结果的平均值。

4.4.1　测试桩复杂度

更低的测试桩复杂度意味着更小的测试代价。实验比较了本文方法和其他方法生成的类集成测试序列的测试桩复杂度。实验结果如图 2 所示，图中横轴表示实验对象，即 ATM，ANT，SPM，DNS，BCEL，JBoss，JHotDraw 和 MyBatis 8 个系统，纵轴表示 GA 方法、RIA 方法、PSO 方法和 DPSO 方法所得的类集成测试序列的测试桩复杂度。

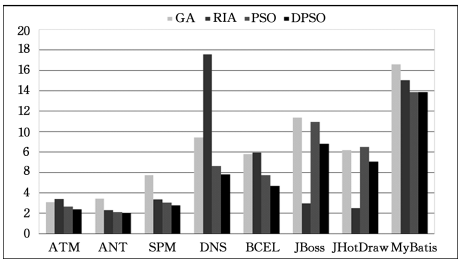


图 2　测试桩复杂度对比图

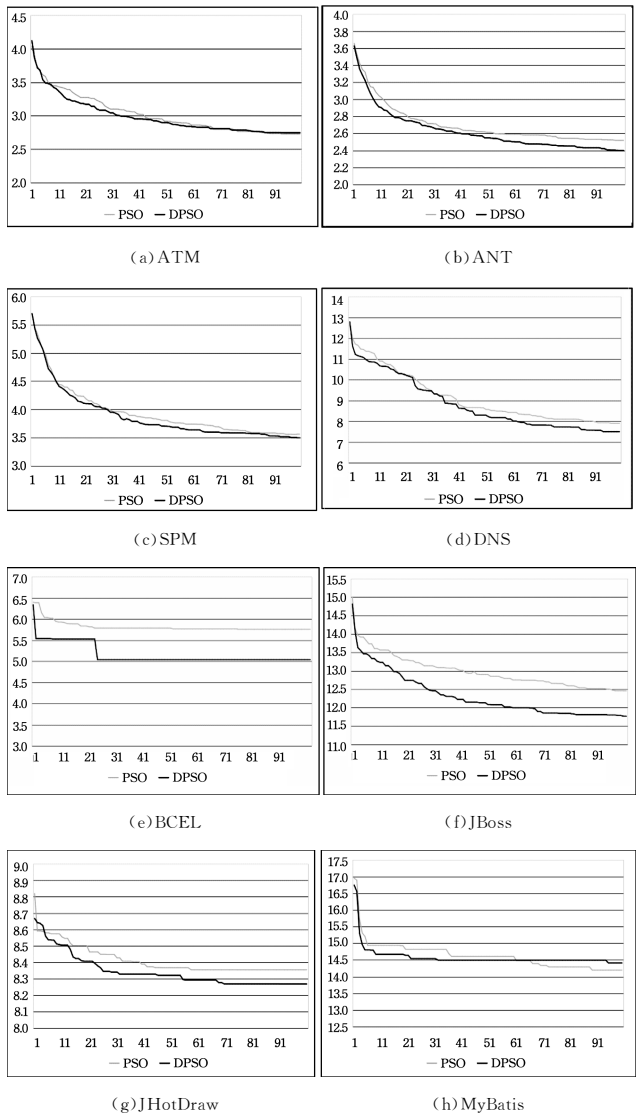
Fig. 2　Comparison results of *OCplx* by different approaches

由图 2 可以看出，在 MyBatis 系统中，本文方法和 PSO 方法所生成的类集成测试序列的测试桩复杂度相同，均小于 GA 方法和 RIA 方法。除此之外，在其余 7 个系统中，本文方法生成的类集成测试序列的测试桩复杂度都比 PSO 方法有不同程度的降低，证明本文方法对 PSO 方法的优化改进是有效的。其中，由于 ATM，ANT，SPM 3 个系统的规模较小，在较小的解空间中，PSO 方法可以找到最优解，本文方法不能充分发挥其优势，但仍优于 GA 方法和 RIA 方法。而对于较大规模的系统 DNS，BCEL，JBoss 和 JHotDraw，本文方法相较于 PSO 方法的优势比较明显，这是因为类的个数较多从而导致解空间大，DPSO 能充分发挥其局部搜索能力强的优势，在每次迭代中可以在当前最优位置附近搜索，避免过早陷入局部最优，从而找到更优的位置。对于 JBoss 和 JHotDraw 两个系统，本文方法生成的类集成测试序列的测试桩复杂度比 GA 方法和 PSO 方法低，但高于 RIA 方法生成的测试序列的测试桩复杂度，原因在于基于搜索的方法采用随机策略初始化种群，导致后代进化存在盲目性，RIA 方法的结果更好是因为其通过模拟退火算法加强了局部搜索能力。

总体上，在 JBoss 和 JHotDraw 中，本文方法的效果仅次于 RIA 方法，而在其他 6 个系统中本文方法生成的类集成测试序列的测试桩复杂度最低。

4.4.2　收敛速度

为了观察本文方法相比 PSO 方法是否可以避免粒子过早陷入局部最优，绘制 8 个系统中本文方法和 PSO 方法的局部最优值(类序列的测试桩复杂度)随迭代次数变化的折线图，取前 100 次迭代的数据，如图 3 所示。其中横轴表示迭代次数，纵轴表示局部最优值。



(a)ATM　(b)ANT

(c)SPM　(d)DNS

(e)BCEL　(f)JBoss

(g)JHotDraw　(h)MyBatis

图 3　DPSO 和 PSO 在 8 个系统上的性能表现

Fig. 3　Performance of DPSO and PSO on eight systems

可以看出，除了 SPM 系统因规模小导致效果不明显以

外，在其余系统上本文方法相较于 PSO 方法可以在迭代初期渐进地对最优值进行搜索，减缓了收敛速度，不容易陷入局部最优。

4.4.3 运行时间

针对不同方法，统计它们各执行一次的平均消时，结果如图 4 所示。其中，横轴表示实验对象，纵轴表示 GA 方法、RIA 方法、PSO 方法以及 DPSO 方法各执行一次的平均耗时。

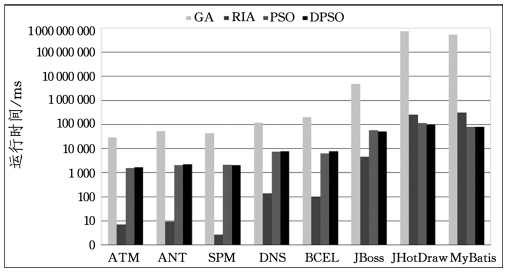


图 4 运行时间对比图

Fig. 4 Comparison of cost time

宏观上，随着系统规模的增大，方法运行时间也随之增加。因为类是构成测试序列的基本元素，所以运行时间与类的数量成正相关。8 个系统中，RIA 方法的运行时间最短，时间优势明显。除此之外，本文方法和 PSO 方法在不同规模的系统下的运行时间差别不大，基本可忽略，且明显少于 GA 方法，GA 方法的运行时间最长。

综上，3 个研究问题得以解决。相较于 GA 方法、RIA 方法和 PSO 方法，本文方法在多数情况下得到的类集成测试序列的测试桩复杂度更低，能减小测试代价；在收敛速度方面，与 PSO 方法相比，本文方法可以减缓收敛速度，避免过早陷入局部最优；在运行时间方面，RIA 方法耗时最短，GA 方法耗时最长。本文方法相较于 PSO 方法在时间消耗上并无明显增加，具有可行性。

4.5 效力分析

本文方法的内部效力威胁在于，尽管把梦境参数 η 设置为粒子的最大速度 V_{MAX}，但由于粒子的做梦能力服从高斯分布，对于一部分粒子来说，Δx 的数量级和 $x_i(t)$ 的数量级仍然相差悬殊，此时在式(7)中扭曲值对粒子位置的偏移作用便微乎其微，仍在一定程度上削弱了方法的效力。此问题并不在于算法本身，而在于算法与实际问题的结合尚有不完善之处。为了减小此威胁，需要在未来的工作中进行更多的实验和改进，使得 DPSO 算法更好地应用于类集成测试序列生成问题。

本文方法的外部效力的威胁在于方法的推广。本文方法取得了较为不错的结果，但不能保证实验结论可推广到其他项目。本文实验对象均来自公认的测试数据集，规模有大有小，代表性较强，可以最大程度地减小威胁。更大范围的推广还需不同领域、不同规模和不同语言(如 C++)的程序来验证。

结束语 本文提出一种基于梦境粒子群优化算法的类集成测试序列确定方法。该方法通过夜间阶段扭曲粒子的自身位置，让粒子有机会在当前最优位置附近搜索，减缓收敛速度，从而得到更好的解，在一定程度上解决了粒子群算法早熟的问题。实验结果表明，在多数情况下本文方法生成的类集成测试序列的测试代价更小。

同时，该方法仍存在一些不足。首先，本文方法对于规模较小的系统优势不明显。其次，本文方法对 PSO 方法早熟问题的解决程度如何以及当参数(如梦境参数 η)设置不同时能否取得更好的结果还需在未来的工作中探索。

参 考 文 献

[1] BRIAND L C, FENG J, LABICHE Y. Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders [C]// Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering. Ischia, Italy, 2002: 43-50.

[2] BRIAND L C, FENG J, LABICHE Y. Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders [R]. Canada: Carleton University, Technical Report: SCE-02-03, 2002.

[3] WANG Z S, LI B X, Wang L L, et al. Using Coupling Measure Technique and Random Iterative Algorithm for Inter-Class Integration Test Order Problem [C]// Proceedings of the 34th Annual IEEE Computer Software and Applications Conference Workshops. Seoul Korea, 2010: 329-334.

[4] ZHANG Y M, JIANG S J, CHEN R Y, et al. Class Integration Testing Order Determination Method Based on Particle Swarm Optimization Algorithm [J]. Chinese Journal of Computers, 2016, 39(55): 1-18. (in Chinese)
张艳梅，姜淑娟，陈若玉，等. 基于粒子群优化算法的类集成测试序列确定方法[J]. 计算机学报，2016，39(55)：1-18.

[5] WANG S S, CHEN M. Dream Effected Particle Swarm Optimization Algorithm [J]. Journal of Information & Computational Science, 2014, 11(15): 5631-5640.

[6] KUNG D C, GAO J, HSIA P, et al. Class Firewall, Test Order, and Regression Testing of Object-Oriented Programs [J]. Journal of Object Oriented Programming, 1993, 8(2): 51-65.

[7] KUNG D C, GAO J, HSIA P, et al. A Test Strategy for Object-Oriented Programs [C]// Proceedings of the 9th International Annual International Computer Software and Applications Conference. Dallas, Texas, USA, 1995: 239-244.

[8] KUNG D C, GAO J, HSIA P, et al. On Regression Testing of Object Oriented Programs [J]. Journal of Systems Software, 1996, 32(1): 21-40.

[9] TAI K C, DANIELS F J. Test Order for Inter-Class Integration Testing of Object-Oriented Software [C]// Proceedings of the 21th International Computer Software and Applications Conference. Washington, USA, 1997: 602-607.

[10] LE TRAON Y, JERON T, MOREL P. Efficient Object-Oriented Integration and Regression Testing [J]. IEEE Transactions on Reliability, 2000, 49(1): 12-25.

[11] BRIAND L C, LABICHE Y, WANG Y. Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles [C]// Proceedings of the 12th International Symposium on Software Reliability Engineering. Hong Kong, China, 2001: 287-296.

[12] BRIAND L C, LABICHE Y, WANG Y. An Investigation of

Graph-Based Class Integration Test Order Strategies [J]. IEEE Transactions on Software Engineering, 2003, 29(7): 594-607.

[13] TARJAN R. Depth-First Search and Linear Graph Algorithms [J]. Siam J of Computing, 2003, 1(4): 144-121.

[14] ZHANG Y M, JIANG S J, ZHANG H C. An Approach for Class Integration Testing Based on the Dynamic Dependency Relations [J]. Chinese Journal of Computers, 2011, 34(6): 1075-1089. (in Chinese)
张艳梅，姜淑娟，张红昌. 一种基于动态依赖关系的类集成测试方法[J]. 计算机学报，2011，34(6)：1075-1089.

[15] ZHANG Y M. Research on Testing Technology of Object-Oriented Programs Based on Dependency Analysis [D]. Xuzhou: China University of Mining and Technology, 2012. (in Chinese)
张艳梅. 基于依赖性分析的面向对象程序测试技术研究[D]. 徐州：中国矿业大学，2012.

[16] JIANG S J, ZHANG Y M, LI H Y, et al. An Approach for Inter-Class Integration Test Order Determination Based on Coupling Measures [J]. Chinese Journal of Computers, 2011, 34(6): 1062-1074. (in Chinese)
姜淑娟，张艳梅，李海洋，等. 一种基于耦合度量的类间集成测试序的确定方法[J]. 计算机学报，2011，34(6)：1062-1074.

[17] HEWETT R, KIJSANAYOTHIN P. Automated Test Order Generation for Software Component Integration Testing [C]// Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 211-220.

[18] ZHANG M, JIANG S J, ZHANG Y M, et al. A Multi-Level Feedback Approach for the Class Integration and Test Order Problem [J]. The Journal of Systems and Software, 2017, 133 (2017): 54-67.

[19] HANH V L, AKIF K, TRAON Y L, et al. Selecting an Efficient OO Integration Testing Strategy: An Experimental Comparison of Actual Strategies [C]// Proceedings of the 15th European Conference on Objecte-Oriented Programming. Budapest, Hungary, 2001: 381-401.

[20] BORNER L, PAECH B. Integration Test Order Strategies to Consider Test Focus and Simulation Effort [C]// Proceedings of the International Conference on Advances in System Testing and Validation Lifecycle. Porto, Portugal, 2009: 80-85.

[21] CABRAL R D V, POZO A, VERGILIO S R. A Pareto Ant Colony Algorithm Applied to the Class Integration and Test Order Problem [C]// Proceedings of the 22th IFIP International Conference on Testing Software and Systems. Natal, Brazil, 2010: 16-29.

[22] VERGILIO S R, POZO A, CABRAL R D V, et al. Multi-Objective Optimization Algorithms Applied to the Class Integration and Test Order Problem [J]. International Journal on Software Tools for Technology Transfer, 2012, 14(4): 461-475.

[23] ASSUNÇÃO W K G, COLANZI T E, POZO A T R, et al. Establishing Integration Test Orders of Classes with Several Coupling Measures [C]// Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation. New York, USA, 2011: 1867-1874.

[24] DEB K, PRATAP A, AGARWAL S, et al. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II [J]. IEEE Transactions on Evolutionary Computation, 2002, 6(2): 182-197.

[25] ZITZLER E, LAUMANNS M, THIELE L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm [R]. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001

[26] KNOWLES J D, CORNEL D W. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy [J]. Evolutionary Computation, 2000, 8(2): 149-172.

[27] GUIZZO G, FRITSCHE G M, VERGILIO S R, et al. A Hyper-Heuristic for the Multi-Objective Integration and Test Order Problem [C]// Proceedings of the Genetic and Evolutionary Computation Conference. Madrid, Spain, 2015: 1343-1350.

[28] GUIZZO G, VERGILIO S R, POZO T R, et al. Grammatical Evolution for the Multi-Objective Integration and Test Order Problem [C]// Proceedings of the Genetic and Evolutionary Computation Conference. Paraná, Brazil, 2016: 1069-1076.

[29] JAROENPIBOONKIT J, SUWANNASART T. Finding a Test Order Using Object-Oriented Slicing Technique [C]// Proceedings of the 20th Asia-Pacific Software Engineering Conference. 2007: 49-56.

[30] LIU L Y. Research of Object-Oriented Software Integration Testing Strategy [D]. Beijing: Beijing University of Posts and Telecommunications, 2013 (in Chinese)
刘颖莲. 面向对象软件集成测试策略研究[D]. 北京：北京邮电大学，2013.

[31] ZHAO Y L, WANG Y, YU H, et al. An Inter-Class Integration Test Order Generation Method Based on Complex Networks [J]. Journal of Northeastern University (Natural Science), 2015, 36(12): 1696-1700. (in Chinese)
赵玉丽，王莹，于海，等. 基于复杂网络的类间集成测试序列生成方法[J]. 东北大学学报(自然科学版)，2015，36(12)：1696-1700.

[32] WANG Y, YU H, ZHU Z L. A Class Integration Test Order Method Based on the Node Importance of Software [J]. Journal of Computer Research and Development, 2016, 53(3): 517-530. (in Chinese)
王莹，于海，朱志良. 基于软件节点重要性的集成测试序列生成方法[J]. 计算机研究与发展，2016，53(3)：517-530.

[33] ZAIDMAN A, DEMEYER S. Automatic Identification of Key Classes in a Software System Using Webmining Techniques [J]. Journal of Software Maintenance and Evolution: Research and Practice, 2008, 20(6): 387-417.

[34] WAMSLEY E J, TUCKER M, PAYNE J D, et al. Dreaming of A Learning Task is Associated With Enhanced Sleep-Dependent Memory Consolidation [J]. Current Biology Cb, 2010, 20(9): 850-855.

[35] HOBSON J A, PACE-SCHOTT E F. The Cognitive Neuroscience of Sleep: Neuronal Systems, Consciousness and Learning [J]. Nature Reviews Neuroscience, 2002, 3(9): 579-693.